

## PATENT ABSTRACTS OF JAPAN

(11)Publication number : 11-134199

(43)Date of publication of application : 21.05.1999

(51)Int.Cl.

G06F 9/45

G06F 9/38

(21)Application number : 09-298150

(71)Applicant : HITACHI LTD

(22)Date of filing : 30.10.1997

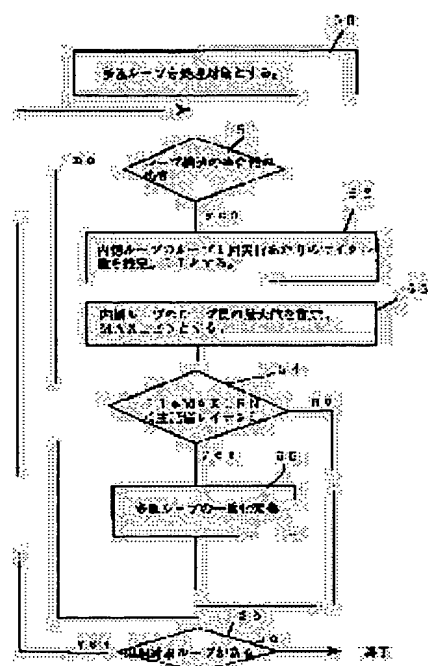
(72)Inventor : TANAKA GIICHI  
TSUSHIMA YUJI  
YASAKA SATOSHI

## (54) PREFETCH CODE GENERATION SYSTEM

(57)Abstract:

**PROBLEM TO BE SOLVED:** To provide an object generation system for hiding main storage penalty when a data area which is accessed by a loop exceeds cache capacity in the multiplex loop.

**SOLUTION:** A code for obtaining a loop length when the dense multiplex loop part of a source program is expressed by a single loop, a code for storing a value when the control variables of the dense multiplex loop are expressed by the single loops in a one-dimensional array and a code for substituted with the one-dimensional array against the reference of the control variable in the multiplex loop are generated. The single loop code of the dense multiplex loop is generated. A prefetch code transferring data necessary for a repetitive operation in the loop after it is made into the single one to a cache when repetitively processing the loop estimated to be previous by data transfer time.



## LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

**THIS PAGE BLANK (USPTO)**

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

**THIS PAGE BLANK (USPTO)**

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平11-134199

(43) 公開日 平成11年(1999) 5月21日

(51) Int.Cl.<sup>6</sup>

G 0 6 F 9/45  
9/38

識別記号

3 3 0

F I

C 0 6 F 9/44  
9/38

3 2 2 C  
3 3 0 E

審査請求 未請求 請求項の数 4 O L (全 11 頁)

(21) 出願番号 特願平9-298150

(22) 出願日 平成9年(1997)10月30日

(71) 出願人 000005108

株式会社日立製作所

東京都千代田区神田駿河台四丁目6番地

(72) 発明者 田中 義一

東京都国分寺市東恋ヶ窪一丁目280番地

株式会社日立製作所中央研究所内

(72) 発明者 對馬 雄次

東京都国分寺市東恋ヶ窪一丁目280番地

株式会社日立製作所中央研究所内

(72) 発明者 家坂 聡

神奈川県横浜市戸塚区戸塚町5030番地 株

式会社日立製作所ソフトウェア開発本部内

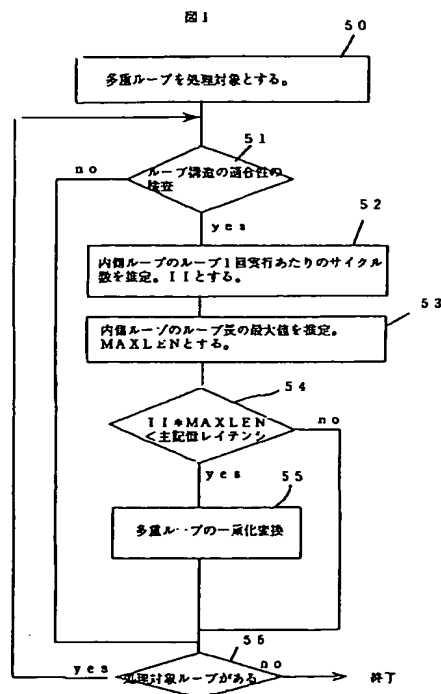
(74) 代理人 弁理士 小川 勝男

(54) 【発明の名称】 プリフェッチコード生成方式

(57) 【要約】

【課題】多重ループにおいて、ループでアクセスするデータ領域がキャッシュ容量を超えた場合に、主記憶ペナリティを隠蔽するオブジェクト生成方式を提供する。

【解決手段】ソースプログラムの密多重ループ部を一重化ループで表現した場合のループ長を求めるコード、上記密多重ループの各々の制御変数を一重ループで表現した場合の値を1次元配列に格納するコード、上記多重ループ内の制御変数参照に対し、上記1次元配列で置換するコードを生成し、上記密多重ループの一重ループコードを生成し、上記一重化後のループ内のある繰り返しの演算に必要なデータを、データ転送時間だけ前と推定されるループの繰り返し処理時に、キャッシュに転送するプリフェッチコードを生成する。



## 【特許請求の範囲】

【請求項1】ソースプログラムをオブジェクトプログラムにコンパイルする方式において、上記ソースプログラムの密多重ループ部を一重化ループで表現した場合のループ長を求めるコード、上記密多重ループの各々の制御変数を一重ループで表現した場合の値を1次元配列に格納するコード、上記多重ループ内の制御変数の参照に対し、上記1次元配列で置換するコードを生成し、上記密多重ループの一重ループコードを生成し、上記一重化後のループ内のある繰り返しの演算に必要なデータを、データ転送時間だけ前と推定されるループの繰り返し処理時に、キャッシュに転送するプリフェッチコードを生成することを特徴とするコード生成方式。

【請求項2】ソースプログラムをオブジェクトプログラムにコンパイルする方式において、上記ソースプログラムの密多重ループ部を一重化ループで表現した場合のループ長がコンパイル時に定数として確定する場合は、上記密多重ループの各々の制御変数を一重ループで表現した場合の値を第1の1次元配列に初期値データとして格納し、上記多重ループ内の制御変数の参照に対し、上記1次元配列で置換するコードを生成し、上記密多重ループの一重ループコードを生成し、上記一重化後のループ内のある繰り返しの演算に必要なデータを、データ転送時間だけ前と推定されるループの繰り返し処理時に、キャッシュに転送するプリフェッチコードを生成することを特徴とするコード生成方式。

【請求項3】請求項2において、上記多重ループ内の配列参照に対するベースアドレスに対する相対アドレスがコンパイル時に確定する場合は、上記相対アドレスを一重ループで表現した場合の値を第2の1次元配列に初期値データとして格納し、上記多重ループ内の配列参照の相対アドレス部を上記第2の1次元配列で置換するコードを生成することを特徴とするコード生成方式。

【請求項4】ソースプログラムをオブジェクトプログラムにコンパイルする方式において、上記ソースプログラムの上記密多重ループ部を一重ループで表現した場合のループ長がコンパイル時に定数として確定しない場合、外側ループの二重ループ化を行い、上記二重ループを外側からループ1、ループ2と呼ぶ時にそれぞれのループ長は、最内側ループの処理をループ2だけ繰り返した推定時間が、データ転送時間のコンパイル時の決めた規定倍以上という指針で決め、ループ2と最内側からなる密多重ループ部を一重化ループで表現した場合のループ長を求めるコード、上記密多重ループの各々の制御変数を一重ループで表現した場合の値を1次元配列に格納するコード、上記多重ループ内の制御変数参照に対し、上記1次元配列で置換するコードを生成し、上記密多重ループの一重ループコードを生成し、上記一重化後のループ内のある繰り返しの演算に必要なデータを、データ転送時間だけ前と推定されるループの繰り返し処理時に、キ

ャッシュに転送するプリフェッチコードを生成することを特徴とするコード生成方式。

## 【発明の詳細な説明】

## 【0001】

【発明の属する技術分野】本発明は、命令レベル並列処理を行うプログラムの実行方式に関わり、プログラムがアクセスするデータが大きく、キャッシュに入りきらない場合のループに好適なコード生成方式に関する。

## 【0002】

【従来の技術】スーパーコンピュータの一つの方向として、スカラプロセッサをノードプロセッサとする並列処理方式が有望視されている。スカラプロセッサを用いた並列スーパーコンピュータが期待されるのは、半導体技術の進歩によるクロック周波数の向上、複数の並列実行可能な演算器を有効に生かすスーパースカラ方式等の命令レベル並列処理の実現により、スカラプロセッサの処理性能が飛躍的に向上しているためである。

【0003】しかしながら、その高いスカラプロセッサ処理能力は、キャッシュが有効に働くときのみ達成される。スカラプロセッサは、一般的に命令処理装置とキャッシュ及びメモリ装置を有する。スカラプロセッサはメモリにプログラムとデータを格納し、プログラムに記述された命令に従いメモリ中のデータを処理する。キャッシュは命令処理装置からの参照時間の短い比較的小容量の記憶手段であり、プログラム及びデータの一部を一時的に格納する。

【0004】命令実行にあたり必要なデータはメモリから読み出されるが、同時にデータを含むデータブロックはキャッシュを構成するラインにコピーされ、以後、当該ブロック内のデータに対する参照が指定されたときは上記キャッシュのラインからデータを参照する。このメモリからキャッシュラインへのデータブロックの転送をライン転送と呼ぶ。命令実行にあたり必要データがキャッシュにない場合、これをキャッシュミスと呼ぶが、キャッシュミスが発生するとライン転送が実行される。

【0005】従来のスカラプロセッサでは命令実行に伴いこのライン転送が発生すると、その完了まで当該命令の実行が待たされる。従って、キャッシュミスが多発するとライン転送に伴う待ち合わせによりプログラムの実行時間が増大し、スカラプロセッサの処理性能が劣化するという問題があった。特に、大規模科学技術計算では、データ領域が大きくデータの局所性が少ないという性質があるため深刻であった。

【0006】これに対し、最近では予めプログラムにおいて、キャッシュミスを引き起こす可能性のある命令に先だってデータの先読みを指示する特殊な命令を実行させることで上記ライン転送に伴う性能劣化を回避する試みがなされている。このデータ先読みをプリフェッチと呼ぶ。例えば、米国IBM社のマイクロプロセッサPowerPCでは指定したオペランドアドレスに位置するデータ

をキャッシュに読み込むプリフェッチ命令がある。

【0007】この動作において、キャッシュミスが発生した場合、プリフェッチ命令の完了を待ち合わせることなくライン転送を行う。従って、上記データを参照する命令を実行する時には上記データがキャッシュに入っているために上記性能劣化が回避される。このようなプリフェッチ命令を利用して、データ転送と演算をオーバーラップさせることで、実質的に主記憶レイテンシを隠蔽させるコード生成技術が特願平9-90470号に示されている。

【0008】

【発明が解決しようとする課題】上記従来のコード生成技術の基本的考え方は、図7のソースプログラムに示すようなDO20(123)及びDO10(124)の多重ループに対するプリフェッチコードの生成において、外側ループDO20のj=j1での内側ループDO10の処理中に、次の外側ループDO20に関する繰り返しj=j1+1で内側ループDO10での処理に必要なデータをプリフェッチすることである。

【0009】しかし、この方式は内側ループのループ長が小さい時に、主記憶レイテンシが十分に隠蔽できない問題があった。例を用いて具体的に示す。

【0010】まず、以下のコード生成で前提としたアーキテクチャの仮定を述べる。対象とするスカラプロセッサはロード/ストア命令が2個、浮動小数点演算が2個、及び整数演算が2個、同時に実行でき、キャッシュラインサイズが32バイト、キャッシュミス時の主記憶ペナルティを100サイクルと仮定する。

【0011】図7のソースプログラムに対して、コンパイラは、まず、ローカリティ解析などによりプリフェッチ対象オペランドを探す。その結果、参照b(i, j)125がプリフェッチ対象オペランドであるとする。オペランドの大きさを8バイトとすると、1回のプリフェッチで、対象とするオペランドを含む連続した32バイトのデータがキャッシュに格納されるため、ループ繰り返しごとにプリフェッチ命令を実行していたのでは無駄で、この例の場合、4回に1回の割合でプリフェッチ命令を出力すればよい。

【0012】プリフェッチコードの生成は以下のように行う。まず、ループDO10のループ1回あたりの実行サイクル数を推定する。ループDO10では、ロード命令が1個、ストア命令が1個、浮動小数点演算が1個で、プリフェッチ命令が1/4であるので、ループ1回あたりの処理サイクル数は $1.125$ サイクル(=max((1+1+1/4)/2, 1/2))と推定できる。キャッシュミス時の主記憶ペナルティは100サイクルであるので、88回(100/1.125=88)前のループ処理で、現在のループ処理で必要となるデータのプリフェッチを行うコードを生成すると、主記憶ペナルティを隠蔽することができる。

【0013】しかし、図7の例において、内側ループDO10のループ長Nが88より小さいときには、従来のコード生成技術では、演算時間が主記憶レイテンシ以下であるので、主記憶レイテンシを完全には隠蔽できない。

【0014】図3は内側ループ長が6の時の実行の様子を模式的に示したものである。横軸が時間、斜線箱がプリフェッチ処理、空箱が演算処理、箱の上には、内側ループ及び外側ループでの処理ループインデックスi, jの値を示してある。ただし、図面の都合上、図7のプログラムとは数値的に一致はしていない(ループ1回あたりの実行サイクル数が1.125サイクルでなく、約10サイクル以上で表示されている)。

【0015】外側ループj=2及び内側ループi=1の演算処理32で必要となるデータを、一つ前の外側ループj=1の内側ループi=1での演算処理30に引き続いてプリフェッチ31を行っているが、内側ループ長が短く、必要な処理時間が短いため、外側ループj=2及び内側ループi=1の演算処理32に必要なデータが到着せずストール33している状態となる。

【0016】内側ループ長が短い場合のコードの性能を見積もる。ループ長がN=30の場合、内側ループを実行するのに、演算処理の実行時間は $1.125 \times 30 = 33.75$ サイクル、主記憶レイテンシが100サイクルであるため、内側ループを実行するのに100サイクルかかり、全体の時間の内66%((100-33.75)/100×100%)は主記憶待ちのオーバーヘッドとなっている。

【0017】さらに、ループ長が短くN=10の場合、内側ループを実行するのに、演算処理の実行時間は $1.125 \times 10 = 11.25$ サイクル、主記憶レイテンシが100サイクルであるため、内側ループを実行するのに100サイクルかかり、全体の時間の内88%((100-11.25)/100×100%)は主記憶待ちのオーバーヘッドとなっている。

【0018】本発明の目的は、上記の問題点を解決するプリフェッチ命令を用いたコード生成方式を提供することにある。

【0019】

【課題を解決するための手段】上記の目的は、ソースプログラムをオブジェクトプログラムにコンパイルする方式において、ソースプログラムの密多重ループ部を一重化ループで表現した場合のループ長を求めるコード、上記密多重ループの各々の制御変数を一重ループで表現した場合の値を1次元配列に格納するコード、上記多重ループ内の制御変数参照に対し、上記1次元配列で置換するコードを生成し、上記密多重ループの一重ループコードを生成し、上記一重化後のループ内のある繰り返しの演算に必要なデータを、データ転送時間だけ前と推定されるループの繰り返し処理時に、キャッシュに転送する

プリフェッチコードを生成することにより達成される。

【0020】

【発明の実施の形態】以下、本発明のコンパイラにおける実施例を図を参照しつつ説明する。

【0021】図2にコンパイラ全体の構造を示す。ソースプログラム1が、字句構文解析2によって中間語3に変換される。ループ構造変換部4は、これを入力として、多重ループに関するプリフェッチコードの主記憶レイテンシの隠蔽される割合を高くするための多重ループに関するループ一重化を行い、中間語5を出力する。最適化部6は、通常の公知の伝統的な最適化を行い、中間語7を出力する。コード生成部8は、プリフェッチ命令を含むオブジェクトコード9を、中間語7をもとに生成する。本発明は4及び8に係わり、オブジェクトコード9の実行効率を向上させるものである。

【0022】図2のループ構造変換部4のうち、多重ループにおける主記憶アクセスペナルティの効率的な隠蔽に関わる部分を図1に示す。図1に入力するソースプログラム1として、図5のFORTRANプログラムを例として説明する。図5のDO20(73)、DO10(74)はループを示し、これらは二重ループを構成している。このようなプログラムに対して図1は以下のような処理を行い、中間語5に変換する。

【0023】図1の処理は、ソースプログラム中の多重ループ50を順次処理する。判定51は、多重ループを考慮したプリフェッチコードを出力する多重ループを選択する。ここでは、密多重ループに限定する。図5のソースプログラムでは、外側ループDO20(73)と内側ループDO10(74)の両者に関係する文75のみが存在するため、この多重ループは密多重ループであるので適合する。

【0024】この例で、外側ループDO20(73)の中に、他のDOループがある場合は、本発明では、多重ループを考慮したプリフェッチ命令は出力しないので、ループ構造が適合しないと判断する。処理52では、元々の内側ループ1回あたりの実行サイクル数を推定する。ループ内には、浮動小数点演算が1個、ロード演算が1個、ストア演算が1個、ラインサイズは32バイトなので、ループ4回に1個のプリフェッチ命令が出力されると推定されるので、ループ1回あたり、 $11 = \max(1+1+1/4)/2$ 、 $1/2 = 1.125$ サイクルである。

【0025】処理53では、最内側のループ長の最大値を推定する。図7の例では、ループ長は変数Nで、コンパイル時には不明であるが、配列宣言122から、言語仕様に従った記述であるかぎり、配列の添字は宣言範囲を超えないと考える。この例では、配列bの1次元目の宣言から、ループ長Nの最大値は50と判定する。判定54は、先に求めたループ1回あたりの実行サイクル数と、最大ループ長から、内側ループの処理実行サイクル

の最大値を求め、主記憶レイテンシと比較し、主記憶レイテンシが大きい場合は、従来技術では主記憶レイテンシの隠蔽は十分でないと判断し、多重ループの一重化変換55へ進む。内側ループの処理実行サイクルが主記憶レイテンシより大きい場合は、従来技術を適用するため、一重化変換を行わない。判定56によりすべての多重ループに対して処理を行う。

【0026】多重ループの一重化変換55の詳細な説明は図4を用いて行う。判定60は一重化後のループ長がコンパイル時に計算可能かを判定する。計算可能な条件は、対象とする多重ループにおいて、外側ループの制御変数の初期値、終値、増分値はコンパイル時に値が確定し(定数)、その他の各ループの制御変数の初期値、終値、増分値は、対象多重ループ内の外側ループの制御変数の関数となることである。図5のDO20(73)、DO10(74)は、この条件を満たす。これに対し、図7のDO20(123)、DO10(124)は、この条件を満たさない。以下、判定60が成立したケースを図5の例、成立しないケースを図7の例を用いて説明する。

【0027】最初に、判定60が成立する場合を説明する。図5の文70～文75は、ソースプログラム1を表しており、字句構文解析部2により、この例の場合、文76～文89のようなコンパイラ内部の中間語3に変換される。ここでは、簡単のために、文79、文82のようにループに関しては、高級言語のままで表現している。tmp\*は整数系の一時変数、ftmp\*は浮動小数点系の一時変数を意味する。

【0028】中間語1fd83は、第2オペランドのアドレス(b(1,1))に、第3オペランドの値(tmp1)を加えた主記憶上のアドレスの8バイト浮動小数点データを第1オペランド(ftmp1)にロードする処理、中間語fadd84は、第2オペランド(ftmp1)と第3オペランド(s)を浮動小数点加算を行い、第1オペランド(ftmp2)に格納する処理、中間語fst85は、第2オペランドのアドレス(a(1,1))に、第3オペランドの値(tmp2)を加えた主記憶上のアドレスに、第1オペランド(ftmp2)の8バイトの浮動小数点データを格納する処理、中間語add86は、第2オペランド(tmp1)と第3オペランド(8)の加算を行い、第1オペランド(tmp1)に格納する処理を表現している。

【0029】ループ内で、tmp1は配列bの原点(b(1,1))、tmp2は配列aの原点(a(1,1))からの相対アドレスを保持している。中間語86、87は、最内側ループにおいて、ループ制御変数が1増えるときの、配列a、bの参照アドレスの増加(8バイト)を表現している。また、中間語88、89は、外側ループのループインデックスが1増えた場合の、配列b、配列aの参照アドレスの増分を表現している。配列aは、配列宣言71から、 $k1 \times 8$ バイト増加し、配列bは、配列宣言72



から400バイト増加する。ここで、k1は、コンパイル時にはわからない変数であることに注意を要する。

【0030】処理61は一重化後のループ長の計算を行う。これは次のように行えばよい。コンパイラ内部で、各ループのループ制御変数の初期値、増分値、終値から同一ループ構造を作成し、内部に計数用の変数cntをおき、実際に計算させればよい。

【0031】

【数1】cnt=0

do j=1, 3

do i=1, j

cnt=cnt+1

end do

上記のループで、変数cntに一重化後のループ長を求めることができ、この例の場合、6となる。

【0032】処理62は、制御変数配列への初期値代入を行う。これは、ループ制御変数を、上記の一重ループで表現したときの値であり、上記の一重ループ化後のループ長を求める時と同様に、次のように求めることができる。

【0033】

【数2】cnt=0

do j=1, 3

do i=1, j

≡IL(cnt+1)=i

≡JL(cnt+1)=j

cnt=cnt+1

end do

ここで、≡ILが内側ループ制御変数の値であり、≡JLは外側ループ制御変数の値である。これを、初期値付データとして扱う。図6の91/92が、求められた初期値付データを表現している。なお、本例では、ループ制御変数を直接的に使うことはなく、8バイトデータの配列のアドレスとしてしか用いられていないので、8倍した値を設定している。

【0034】処理63は、配列のベースアドレスよりの相対アドレスが確定する参照に対して、その値を相対アドレス配列への初期値として代入する。相対アドレスが確定する参照とは、配列bの参照75のように一重化後のループインデックスの値に対して、配列のベースアドレス(b(1,1))からの相対アドレスが定数で決まる場合をいう。配列b(i,j)の相対アドレスは、b(i,j)のアドレスとb(1,1)のアドレスの差が $(i-1+50 \times (j-1)) \times 8$ バイトであることから

【0035】

【数3】cnt=0

do j=1, 3

do i=1, j

≡bind(cnt+1)=(i-1+50\*(j-1))  
\*8

cnt=cnt+1

end do

というプログラムにより求めることができる。ここで、≡bindが参照b(i,j)の相対アドレスデータであり、図6の90が、求められた初期値データを表している。なお、配列参照a(i,j)は、相対アドレスが $(i-1+k1*(j-1)) \times 8$ であるが、変数としてループ制御変数i,j以外のk1(配列aの宣言71の1次元目)を含んでいるために、コンパイル時に定数として計算できない。これに関しては実行時に計算することになる。

【0036】処理64は多重ループ部分のループ一重化変換を行う。DO20(79)とDO10(82)がさきほど求めたループ長が6となる一重ループDO30(97)に置き換わる。配列参照b(i,j)のアドレス計算コードは、77,80,86,88から構成されていたが、一重化により、コード94,98,106に置き換わる。ここでは、配列参照b(i,j)は、処理65により、相対アドレス計算コードが除去され、先に求めた相対アドレス配列≡bindをロード(98)する処理に最適化されている。

【0037】また、配列参照a(i,j)のアドレス計算コードは、76,78,81,87,89から構成されていたが、一重化により、コード93,95,96,101,102,103,104,107,108に置き換わる。ここで、コード101,107が、制御変数≡ILの値、コード102,108が、制御変数≡JLの値を意味している。配列参照b(i,j)の相対アドレス $(i-1+k1*(j-1))$ は、これらから、コード103,104から求めることができる。

【0038】次に、判定60が成立せず、一重化後のループ長がコンパイル時に計算できない場合に関して、図7を用いて説明する。図5のプログラムとの違いは、ループインデックスの範囲(123,124)が異なり、コンパイル時に一重化後のループ長を求めることができないことである。このため、図5の処理においてコンパイル時にに行っていた操作を、コードとして生成し、実行時に求めることになる。

【0039】まず、処理66において外側ループを二つの多重ループ(外側ループ1と外側ループ2)に分解する。分割の基準は、外側ループ2と内側ループを一重化した場合の処理時間と主記憶レイテンシの時間比を例えば20程度とするように決める。この値を大きくすればするほど、主記憶レイテンシのオーバーヘッドが見えなくなるが、一重化のための作業配列容量が大きくなる。

【0040】図7の例の場合、DO20(129)は、図8のDO25(151),DO26(154)の二つに分けられる。ここで、コード145~150及び、コード152~153,155は二重ループ化のために必要なものである。ここでの分割点は次のように求めた。

まず、ループ1回あたりの実行サイクル数を推定する。

【0041】コード133～137の必要な演算資源は、メモリ系命令が2、浮動小数点系命令が1、整数系命令が2であるが、一重化されることによりメモリ参照系命令のアドレス計算コードが増加し、さらに配列参照bに対するプリフェッチコードのためにコードが増加し、結局、メモリ系命令が7、浮動小数点系命令が1、整数系が8となる(図11のコードD042)と推定される。メモリ系命令が2個、浮動小数点系命令が2個、そして整数系命令が2個が同時に実行できる計算機の場合、このループの1回あたりの実行サイクル数は4(=max(7/2, 1/2, 8/2))と推定できる。また、最内側ループ長N(124)の最大値は、配列宣言122より50と推定されるので、最内側ループの処理時間の最大値は200(=4×50)サイクルと推定される。主記憶レイテンシを100サイクルと仮定し、主記憶レイテンシのオーバーヘッドを1/20とするには最内側処理を約10回まわすように外側ループ2を決めればよい。

【0042】処理67、処理68は、外側ループ2と内側ループの一重化後のループ長計算中間語、及び、制御変数配列への初期値代入中間語をコード中に挿入する。図9のコード160～166がこれを意味する。コード167のtmp28に一重化後のループ長が保持される。また、ここで、制御変数配列が新たに導入されたが、コード144のように、最大ループ長500要素分(50(内側ループ最大ループ長)×10(外側ループ2の最大ループ長))を確保する。

【0043】処理69は、外側ループ2と内側ループのループ一重化による中間語変換を行う。図8のD026(154)とD010(132)の含む文のループの一重化を行うことにより、図9のD040(170)のループ一重化が行われる。配列参照b(i, j)のアドレス計算コードは、130, 136, 138から構成されていたが、一重化により、コード168, 169, 171, 172, 173, 174, 180, 181に置き換わる。また配列参照a(i, j)のアドレス計算コードは、131, 137, 139から構成されていたが、一重化により、コード168, 169, 171, 172, 177, 178, 180, 181に置き換わる。両参照に共通であるコード171, 172, 180, 181は、元々のループでの制御変数配列の値のロード処理を意味する。

【0044】その後、通常の間置語に対する最適化部6の処理を行い、コード生成部8においてコード生成を行う。コード生成部8における、多重ループの一重化による主記憶アクセスペナルティの効率的な隠蔽に関わる部分を図10に示す。処理200は、最内側ループ内のプリフェッチ対象ロードオペランドの選択を行う。選択の方法は、例えば、公知技術であるTodd C. Mowry Monica S.

Lam Anoop Gupta Design and Evaluation of a Compiler Algorithm for Prefetching, ASPLOS-V, ACM 0-89791-535-6/92/0010/0062 pp.62-73に従う。この結果、図9のb(i, j)175がプリフェッチ対象オペランドとなるとする。

【0045】処理201では、最内側ループのループ範囲を二つに分割する。後半ループのループ長には、実行時間を主記憶アクセスペナルティ以上にするループ長、か、または、ループ長が短く主記憶アクセスペナルティ以上の実行時間にならないときは、もともとのループ長を設定する。前半ループのループ長はその残りとする。従って、前半ループは実行されないこともある。図9のD040(170)のループ1回あたりの実行推定サイクルは前に述べたようにMINL=4である。このため、主記憶レイテンシを100とすると、後半ループのループ長は、tmp50=min(tmp28, 100/4)となる。その結果、図9のD040(170)は、図11のループ長tmp51(223)の前半ループD042(232)、図12のループ長tmp50(222)の後半ループD043(251)に分割できる。

【0046】処理202は、前半ループで必要となるプリフェッチコード生成を行う。図11のD041(224)が対応するコードである。これは、配列参照b(i, j)のアドレス計算コード171, 172, 173, 174, 180, 181をコピーして一時変数をリネームすれば得られる。すなわち、tmp35をtmp65, tmp36をtmp66, tmp37をtmp67, tmp38をtmp68, tmp30をtmp80, tmp31をtmp81にリネームすればよい。後者の番号を用いたコード系列がプリフェッチを生成するためのアドレス計算を意味する。コード220, コード221は、リネームした一時変数の初期値の設定である。このループのループ長は、tmp50で、上記の二分割ループの後半のループと同一である。また、prefetch229は、第1オペランド(b(1, 1))のアドレスに第2オペランド(tmp38)を加算したアドレスを含む1ラインのデータを主記憶からキャッシュに転送する中間語を意味する。

【0047】処理203は、上記前半ループのコード生成を行う。ここでは、もともとのループボディ(コード171～181)に、処理201と同一のプリフェッチのためのコード(233～237, 238, 239)を挿入したものである。これは、ループi回目において、ループ(i+100/4)回で必要となるデータをプリフェッチしていることになる。

【0048】処理204は、上記の後半ループに対するコード生成を行う。コード171～181とループ長を除いてまったく同一である。

【0049】そして、実際の機械語を生成し、オブジェクトコード9を生成する。

【0050】

【発明の効果】本発明のコード生成方式により、多重ル

ープにおいて、ループを一重化して、必要なデータを主記憶レイテンシの時間に相当するループ繰り返し前にプリフェッチを開始するため、主記憶レイテンシを隠蔽することができる。図7の例で内側ループNが10で、外側ループ長Mが10の時の処理時間を推定する。従来方式では、先に述べたように、内側ループ長が小さいときには、図3のように、内側ループの処理の完了毎にデータ待ちのストールが発生するため、1000サイクル（100（内側ループ処理時間）×10（外側ループ長））となる。

【0051】本方式によれば、内側ループと外側ループの全体の長さが長ければ、図13のように、主記憶ベンラティは最初を除き軽減することができる。このため、本方式のコードではループ1回あたり4サイクルの時間がかかることから、ループ処理での400=4×10（内側ループ長）×10（外側ループ長）とループ最初のデータの待ちの100サイクルのオーバヘッドを合わせて500サイクルとなる。この性能は従来方式に比べて約2倍の性能である。

【図面の簡単な説明】

【図1】本発明による多重ループ処理方式を示すフロー図。

【図2】本発明のコンパイラの全体の構成と機能を示す説明図。

【図3】従来法による実行の様子を示した説明図。

【図4】本発明によるループ一重化方式のフロー図。

【図5】本発明による多重ループの一例のプログラムを示す図。

【図6】本発明による多重ループの一重化変換の一例のプログラムを示す図。

【図7】本発明による多重ループの他の例のプログラムを示す図。

【図8】本発明による多重ループの一重化変換の他の例のプログラムを示す図。

【図9】本発明による多重ループの一重化変換の他の例のプログラムを示す図。

【図10】本発明によるプリフェッチコード生成方式を示すフロー図。

【図11】本発明による多重ループのプリフェッチ変換の一例のプログラムを示す図。

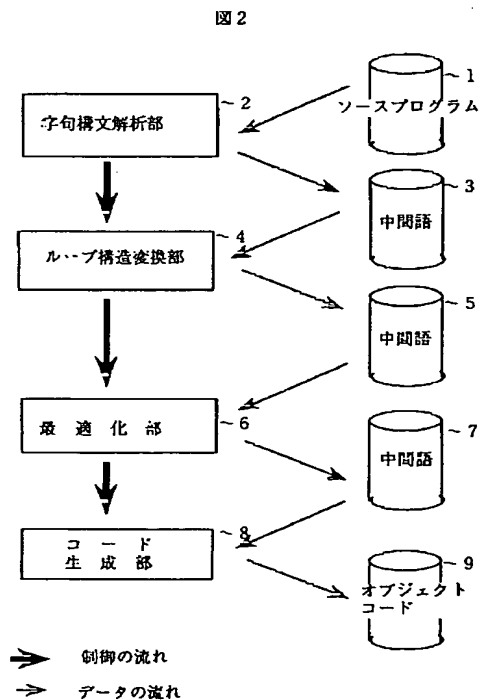
【図12】本発明による多重ループのプリフェッチ変換の他の例のプログラムを示す図。

【図13】本発明による実行の様子を示した説明図。

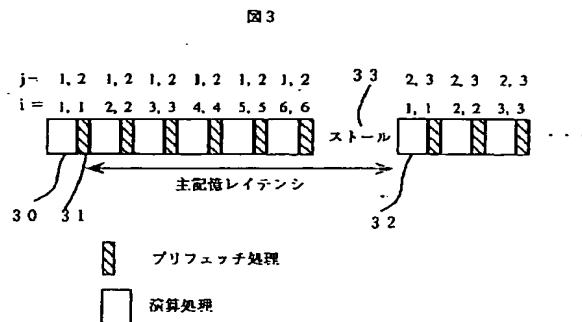
【符号の説明】

1…ソースプログラム、3…字句構文解析、3…中間語、4…ループ構造変換部、5…中間語、6…最適化部、7…中間語、8…コード生成部、9…プリフェッチ命令を含むオブジェクトコード。

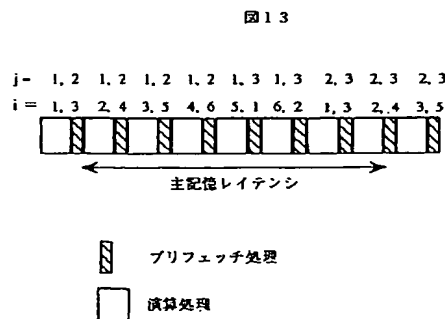
【図2】



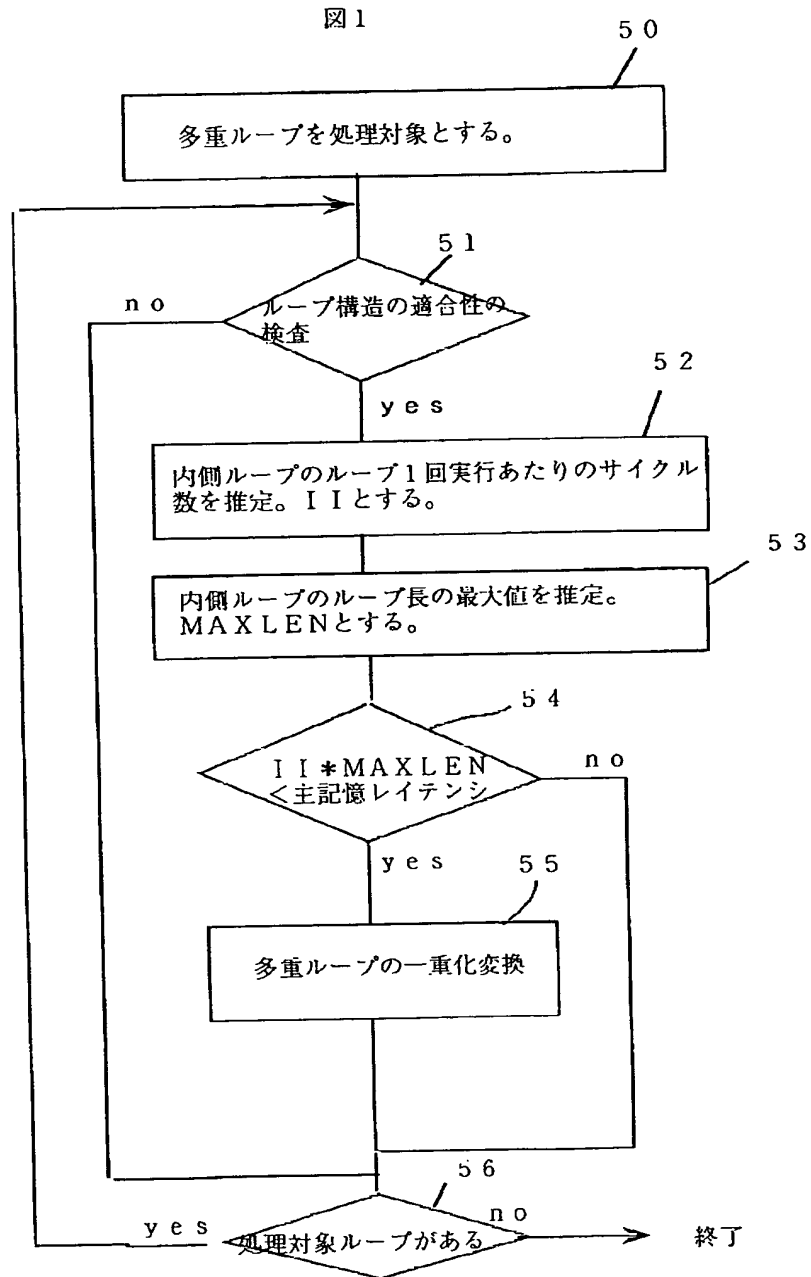
【図3】



【図13】

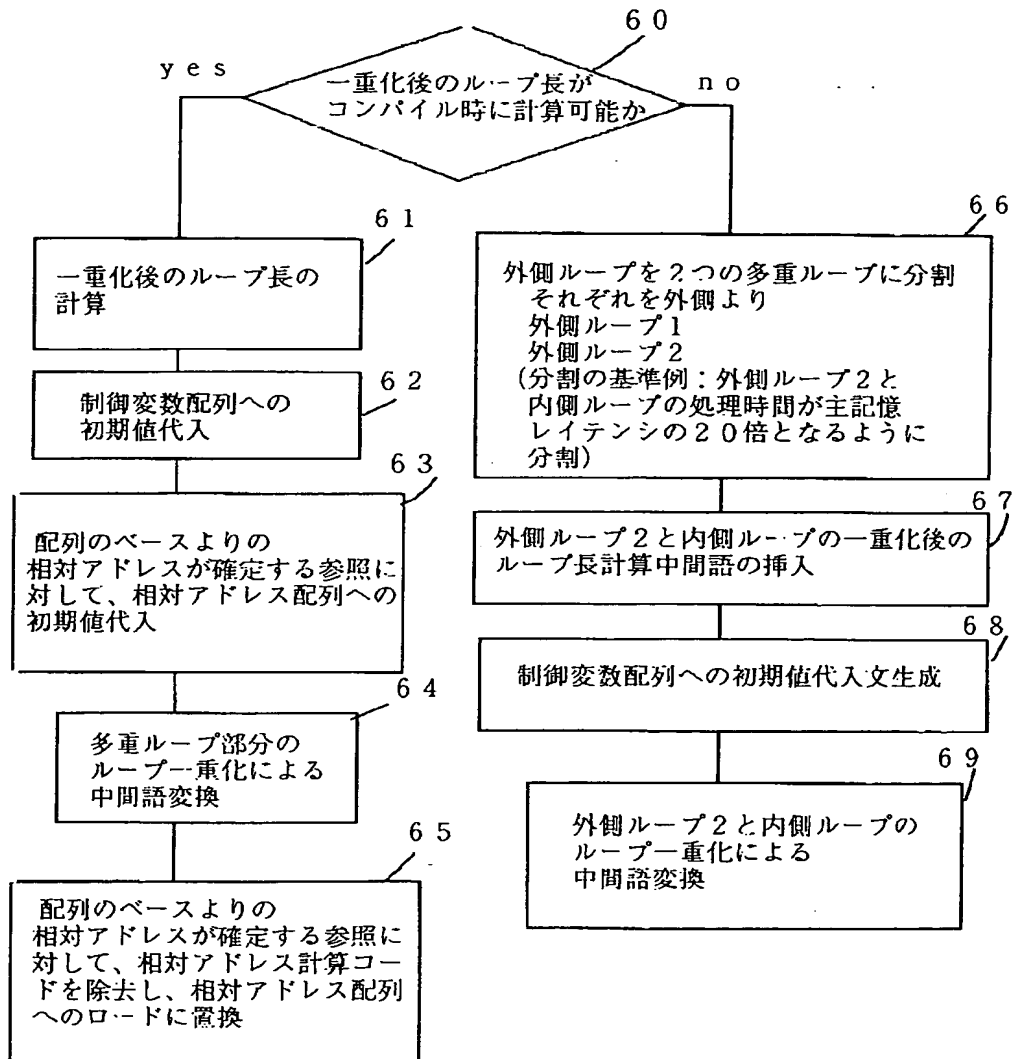


【図1】



【図4】

図4



【図5】

図5

```

subroutine xxx(a, k1)  — 70
real*8 a(k1, 100)    — 71
real*8 b(50, 100)    — 72

DO 20 j=1, 3          — 73
DO 10 i=1, j          — 74
    a(i, j) = b(i, j) + s  — 75

10 continue
20 continue

    ↓↓

    tmp5=8*k1          — 76
    tmp3=0             — 77
    tmp4=0             — 78

DO 20 j=1, 3          — 79
    tmp1=tmp3          — 80
    tmp2=tmp4          — 81

DO 10 i=1, j          — 82
    lfd ftmp1, b(1, 1), tmp1 — 83
    fadd ftmp2, ftmp1, s — 84
    fst ftmp2, a(1, 1), tmp2 — 85

    add tmp1, tmp1, =8 — 86
    add tmp2, tmp2, =8 — 87

10 continue

    add tmp3, tmp3, =8*50 — 88
    add tmp4, tmp4, tmp5 — 89

20 continue

```

【図7】

図7

```

subroutine xxx(a, k1)  — 120
real*8 a(k1, 100)    — 121
real*8 b(50, 100)    — 122

DO 20 j=1, M          — 123
DO 10 i=1, N          — 124
    a(i, j) = b(i, j) + s  — 125

10 continue
20 continue

    ↓↓

    tmp5=8*k1          — 126
    tmp3=0             — 127
    tmp4=0             — 128

DO 20 j=1, M          — 129
    tmp1=tmp3          — 130
    tmp2=tmp4          — 131

DO 10 i=1, N          — 132
    lfd ftmp1, b(1, 1), tmp1 — 133
    fadd ftmp2, ftmp1, s — 134
    fst ftmp2, a(1, 1), tmp2 — 135

    add tmp1, tmp1, =8 — 136
    add tmp2, tmp2, =8 — 137

10 continue

    add tmp3, tmp3, =8*50 — 138
    add tmp4, tmp4, tmp5 — 139

20 continue

```

【図6】

図6

主記憶領域

```

%bind /0.400, 408, 800, 808, 816/ — 90
%ll /8, 8, 16, 8, 16, 24/ — 91
%jl /8, 16, 16, 24, 24, 24/ — 92

    tmp13=k1          — 93
    tmp7=0            — 94
    tmp8=0            — 95
    tmp9=0            — 96

DO 30 ii=1, 6        — 97
    lfd tmp6, %bind(1), tmp7 — 98
    lfd ftmp1, b(1, 1), tmp6 — 99
    fadd ftmp2, ftmp1, s — 100

    lfd tmp10, %ll(1), tmp8 — 101
    lfd tmp11, %jl(1), tmp9 — 102
    mul tmp12, tmp11, tmp13 — 103
    add tmp14, tmp10, tmp12 — 104
    fst ftmp2, a(1, 1), tmp14 — 105

    add tmp7, tmp7, =4 — 106
    add tmp8, tmp8, =4 — 107
    add tmp9, tmp9, =4 — 108

30 continue

```

【図8】

図8

```

    tmp5=8*k1          — 126
    tmp3=0             — 127
    tmp4=0             — 128

    tmp20=10           — 145
    tmp25=tmp20        — 146
    tmp21=M/tmp20      — 147
    tmp22=M-tmp21*tmp20 — 148
    if (tmp22, ne, 0) then — 149
        tmp21=tmp21+1 — 150
    endif

DO 25 j1=1, tmp21     — 151
    if (j1, eq, tmp21, and, — 152
        tmp22, ne, 0) then
        tmp25=tmp22 — 153
    endif

DO 26 j2=1, tmp25     — 154
    j=j2+tmp20*(j1-1) — 155

    tmp1=tmp3          — 130
    tmp2=tmp4          — 131

DO 10 i=1, N          — 132
    lfd ftmp1, b(1, 1), tmp1 — 133
    fadd ftmp2, ftmp1, s — 134
    fst ftmp2, a(1, 1), tmp2 — 135

    add tmp1, tmp1, =8 — 136
    add tmp2, tmp2, =8 — 137

10 continue

    add tmp3, tmp3, =8*50 — 138
    add tmp4, tmp4, tmp5 — 139

26 continue
25 continue

```

【図9】

図9

```

Dimension YIL (500), ¥JL (500)  -144

tmp20=10 -145
tmp25=tmp20 -146
tmp21=M/tmp20 -147
tmp22=M-tmp21*tmp20 -148
if (tmp22.ne.0) then -149
  tmp21=tmp21+1 -150
endif

DO 25 j1=1, tmp21 -151
  if (j1.eq.tmp21.and. -152
    tmp22.ne.0) then -153
    tmp25=tmp22 -153
  endif

  tmp27=1 -160
DO 30 j2=1, tmp25 -161
DO 31 i=1, N -162
  j=j2+tmp20*(j1-1) -163
  ¥IL (tmp27)=j*8 -164
  ¥JL (tmp27)=j*8 -165
  tmp27=tmp27+1 -166
31 continue
30 continue

  tmp28=tmp27-1 -167
  tmp30=0 -168
  tmp31=0 -169

DO 40 ii=1, tmp28 -170
  l tmp35, ¥IL (1), tmp30 -171
  l tmp36, ¥JL (1), tmp31 -172
  mul tmp37, tmp36, =50 -173
  add tmp38, tmp35, tmp37 -174
  lfd ftmp10, b(1,1), tmp38 -175
  fadd ftmp11, ftmp10, s -176
  mul tmp38, tmp36, k1 -177
  add tmp39, tmp35, tmp38 -178
  fst ftmp11, a(1,1), tmp39 -179
  add tmp30, tmp30, =4 -180
  add tmp31, tmp31, =4 -181

40 continue
25 continue

```

【図12】

図12

```

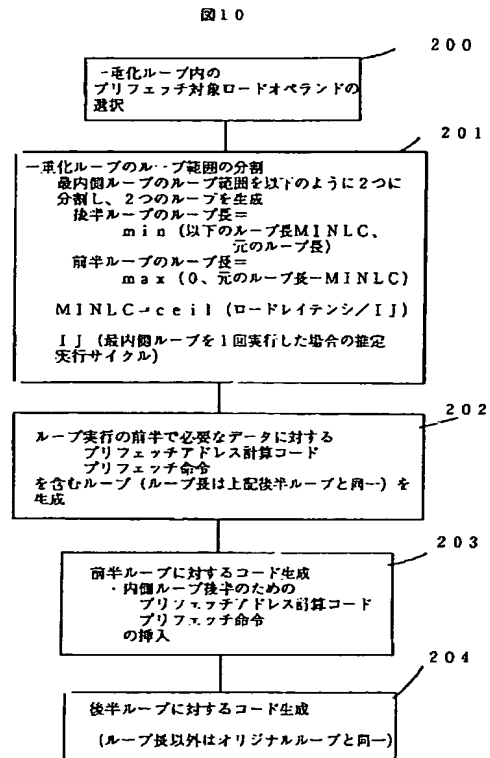
DO 43 ii=1, tmp50 -251

  l tmp35, ¥IL (1), tmp30 -252
  l tmp36, ¥JL (1), tmp31 -253
  mul tmp37, tmp36, =50 -254
  add tmp38, tmp35, tmp37 -255
  lfd ftmp10, b(1,1), tmp38 -256
  fadd ftmp11, ftmp10, s -257
  mul tmp38, tmp36, k1 -258
  add tmp39, tmp35, tmp38 -259
  fst ftmp11, a(1,1), tmp39 -260
  add tmp30, tmp30, =4 -261
  add tmp31, tmp31, =4 -262

43 continue

```

【図10】



【図11】

図11

```

tmp30=0 -168
tmp31=0 -169
tmp80=tmp30 -220
tmp81=tmp31 -221

tmp50=min (100/4, tmp28) -222
tmp51=tmp28-tmp50 -223

DO 41 ii=1, tmp50 -224
  l tmp65, ¥IL (1), tmp80 -225
  l tmp66, ¥JL (1), tmp81 -226
  mul tmp67, tmp66, =50 -227
  add tmp68, tmp65, tmp67 -228
  prefetch b(1,1), tmp38 -229
  add tmp80, tmp80, =4 -230
  add tmp81, tmp81, =4 -231
41 continue

DO 42 ii=1, tmp51 -232
  l tmp65, ¥IL (1), tmp80 -233
  l tmp66, ¥JL (1), tmp81 -234
  mul tmp67, tmp66, =50 -235
  add tmp68, tmp65, tmp67 -236
  prefetch b(1,1), tmp38 -237
  l tmp35, ¥IL (1), tmp30 -171
  l tmp36, ¥JL (1), tmp31 -172
  mul tmp37, tmp36, =50 -173
  add tmp38, tmp35, tmp37 -174
  lfd ftmp10, b(1,1), tmp38 -175
  fadd ftmp11, ftmp10, s -176
  mul tmp38, tmp36, k1 -177
  add tmp39, tmp35, tmp38 -178
  fst ftmp11, a(1,1), tmp39 -179
  add tmp30, tmp30, =4 -180
  add tmp31, tmp31, =4 -181
  add tmp80, tmp80, =4 -238
  add tmp81, tmp81, =4 -239
42 continue

```

THIS PAGE BLANK (USPTO)